Informe de Laboratorio 16
Tema: Programación Orientada a Objetos (III)

Nota

Estudiante	Escuela	Asignatura
Roni Companocca Checco rcompanocca@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Semestre: II Código: 20210558

Laboratorio	Tema	Duración
16	Programación Orientada a Objetos (III)	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 20 Noviembre 2023	Al 25 Noviembre 2023

1. TAREA

1.1. Objetivos:

- Que el alumno demuestre poder crear "clases definidas por el programador"
- Implementar métodos para las clases definidas por el programador
- Utilizando atributos que son otros objetos. Agregación, composición, herencia.

1.2. Competencias a alcanzar:

- Diseña, responsablemente, sistemas, componentes o procesos para satisfacer necesidades dentro de restricciones realistas: económicas, medio ambientales, sociales, políticas, éticas, de salud, de seguridad, manufacturación y sostenibilidad.
- Aplica de forma flexible, técnicas, métodos, principios, normas, estándares y herramientas de ingeniería necesarias para la construcción de software e implementación de sistemas de información.

2. EQUIPOS, MATERIALES Y TEMAS UTILIZADOS

- Sistema Operativo Windows
- OpenJDK 64-Bits 17.0.7.
- Git 2.39.2.
- Cuenta en GitHub con el correo institucional.

3. URL DE REPOSITORIO GITHUB

- URL para el Repositorio GitHub.
- `https://github.com/RONI-COMPANOCCA-CHECCO`
- URL para el laboratorio 16 en el Repositorio GitHub.
- `https://github.com/RONI-COMPANOCCA-CHECCO/FP2-LAB-16`

4. EJERCICIO RESUELTO

EJERCICIO 1: Crear un POO donde se implementa la herencia y el polimorfismo.

1.1 Creamos la Clase Padre Figura:

```
public abstract class Figura {
    public static int contadorFiguras = 1;
    //ATRIBUTOS
    private int posicionX;
    private int posicionY;
    protected int id;
    //METODOS GETTER Y SETTER
    public int getPosicionX(){
        return posicionX;
    }
    public int getPosicionY(){
        return posicionY;
    }
    public void setPosicionX(int posicionX){
        this.posicionX = posicionX;
    }
    public void setPosicionY(int posicionY){
        this.posicionY = posicionY;
    }
    public int getId(){
        return id;
    }
    public void setId(int id){
        this.id = id;
    }
    //METODOS ABSTRACTOS
    public abstract double calcularArea();
    public abstract void dibujar();
}
```

1.2 Creamos la Clase Hija Cuadrado:

```
public class Cuadrado extends Figura{
    //ATRIBUTOS
    private double lado;
    //CONSTRUCTORES
    public Cuadrado(double lado){
        this.lado = lado;
        this.id = Figura.contadorFiguras++;
```

```
    }
    //METODOS GETTER Y SETTER
    public double getLado(){
        return lado;
    }
    public void setLado(double lado){
        this.lado = lado;
    }
    //METODOS PUBLICOS
    public double calcularArea(){
        double resultado;
        resultado = lado*lado;
        return resultado;
    }
    public void dibujar(){
        System.out.println("Dibujando CUADRADO "+ this.id);
    }
}
```

1.3 Creamos la Clase Hija Círculo:

```
public class Circulo extends Figura{
    //ATRIBUTOS
    private double radio;
    private final double PI = 3.14159;
    //CONSTRUCTORES
    public Circulo(double radio){
        this.radio = radio;
        this.id = Figura.contadorFiguras++;
    }
    //METODOS SETTER Y GETTER
    public double getRadio(){
        return radio;
    }
    public void setRadio(double radio){
        this.radio = radio;
    }
    //METODOS PUBLICOS
    public double calcularArea(){
        double resultado;
        resultado = PI *radio*radio;
        return resultado;
    }
    public void dibujar(){
        System.out.println("Dibujando CIRCULO: "+this.id);
    }

}
```

1.4 Creamos la Clase Hija Triángulo:

```
public class Triangulo extends Figura{
    //ATRIBUTOS
    private double base;
```

```
    private double altura;
    //METODOS GETTER Y SETTER
    public double getBase(){
        return base;
    }
    public void setBase(double base){
        this.base = base;
    }
    public double getAltura(){
        return altura;
    }
    public void setAltura(double altura){
        this.altura = altura;
    }
    //CONSTRUCTORES
    public Triangulo(double base, double altura){
        this.base = base;
        this.altura = altura;
        this.id = Figura.contadorFiguras++;
    }
    public double calcularArea(){
        double resultado;
        resultado = base*altura/2;
        return resultado;
    }
    public void dibujar(){
        System.out.println("dibujando TRIANGULO "+this.id);
    }
}
```

1.5 Creamos la Clase Main, donde instanciamos a nuestras clases y llamamos a las operaciones implementadas en las clase hijas, haciendo uso del polimorfismo:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class Main{
    //LISTA DE OBJETOS FIGURA
    static ArrayList<Figura> lienzo = new ArrayList<Figura>();
    static Scanner consola = new Scanner(System.in);
    public int menuFiguras(){
        System.out.println("--------------------------------");
        System.out.println("\tFIGURAS");
        System.out.println("--------------------------------");
        System.out.println("1. Cuadrado");
        System.out.println("2. Circulo");
        System.out.println("3. Triangulo");
        System.out.println("--------------------------------");
        int opcion = consola.nextInt();
        return opcion;
    }
    public void crearFigura(ArrayList<Figura> lienzo, int opc){
        if(opc==1){
            System.out.println("Ingrese el LADO del cuadrado: ");
            int lado = consola.nextInt();
```

```
            lienzo.add(new Cuadrado(lado));
        }
        if(opc==2){
            System.out.println("Ingrese el RADIO del Circulo: ");
            int radio = consola.nextInt();
            lienzo.add(new Circulo(radio));
        }
        if(opc==3){
            System.out.println("Ingrese la BASE del TRIANGULO: ");
            int base = consola.nextInt();
            System.out.println("Ingrese la Altura del TRIANGULO: ");
            int altura = consola.nextInt();

            lienzo.add(new Triangulo(base, altura));
        }
    }
    public void imprimirAreasFiguras(ArrayList<Figura> lienzo){
        for(Figura objFigura: lienzo){
            if(objFigura instanceof Cuadrado)
                System.out.print("Area del Cuadrado "+objFigura.getId()+" : ");
            else
                if(objFigura instanceof Circulo)
                    System.out.print("Area del Circulo "+objFigura.getId()+" : ");
                else
                    if(objFigura instanceof Triangulo){
                        System.out.print("Area del Triangulo "+objFigura.getId()+" :
                            ");
            //POLIMORFISMO
            System.out.println(objFigura.calcularArea());
            }
        }
    }
    public void imprimirFiguras(ArrayList<Figura> lienzo){
        for(Figura objFigura: lienzo){
            //POLIMORFISMO
            objFigura.dibujar();
        }
    }
    public static void main(String[] args){
        int opcion;
        Main principal = new Main();
        do{
            System.out.println(
                "-----------------------------------------\n"+
                "1. Crear figura\n"+
                "2. Imprimir Todas las Figuras\n"+
                "3. Calcular Areas\n"+
                "4. Imprimir Tio de Figuras\n"+
                "5. Imprimir un objeto Figura\n"+
                "6. FIN\n"+
                "-----------------------------------------\n"+
                "Ingrese la opcion (1-6)"
            );
            opcion = consola.nextInt();
            switch(opcion){
                case 1: //
```

```
                    int opc = principal.menuFiguras();
                    principal.crearFigura(lienzo, opc);
                    break;
             case 2: //
                    principal.imprimirFiguras(lienzo);
                    break;
             case 3: //
                    principal.imprimirAreasFiguras(lienzo);
                    break;
             case 0:
                    System.exit(0);
                    break;
          }
       }while (true);
    }
}
```

Ejecucion:

```
-------------------------------------------
1. Crear figura
2. Imprimir Todas las Figuras
3. Calcular Areas
4. Imprimir Tio de Figuras
5. Imprimir un objeto Figura
6. FIN
-------------------------------------------
Ingrese la opcion (1-6)
1
-------------------------------
   FIGURAS
-------------------------------
1. Cuadrado
2. Circulo
3. Triangulo
-------------------------------
1
Ingrese el LADO del cuadrado:
2
-------------------------------------------
1. Crear figura
2. Imprimir Todas las Figuras
3. Calcular Areas
4. Imprimir Tio de Figuras
5. Imprimir un objeto Figura
6. FIN
-------------------------------------------
Ingrese la opcion (1-6)
1
-------------------------------
   FIGURAS
-------------------------------
1. Cuadrado
2. Circulo
3. Triangulo
```

```
------------------------------
2
Ingrese el RADIO del Circulo:
3
------------------------------------------
1. Crear figura
2. Imprimir Todas las Figuras
3. Calcular Areas
4. Imprimir Tio de Figuras
5. Imprimir un objeto Figura
6. FIN
------------------------------------------
Ingrese la opcion (1-6)
2
Dibujando CUADRADO 1
Dibujando CIRCULO: 2
------------------------------------------
1. Crear figura
2. Imprimir Todas las Figuras
3. Calcular Areas
4. Imprimir Tio de Figuras
5. Imprimir un objeto Figura
6. FIN
------------------------------------------
Ingrese la opcion (1-6)
```

5. EJERCICIO PROPUESTO

PROBLEMA 01

En la historia, los ejércitos estaban conformados por diferentes tipos de soldados, que tenían similitudes, pero también particularidades.

Basándose en la clase Soldado crear las clases Espadachín, Arquero y Caballero. Las tres clases heredan de la superclase Soldado, pero aumentan atributos y métodos, o sobrescriben métodos heredados.

Los espadachines tienen como atributo particular "longitud de espadaz como acción çrear un muro de escudos"que es un tipo de defensa en particular.

Los caballeros pueden alternar sus armas entre espada y lanza, además de desmontar (sólo se realiza cuando está montando e implica defender y cambiar de arma a espada), montar (sólo se realiza cuando está desmontado e implica montar, cambiar de arma a lanza y envestir). El caballero también puede envestir, ya sea montando o desmontando, cuando es desmontado equivale a atacar 2 veces, pero cuando está montando implica a atacar 3 veces.

Los arqueros tienen un número de flechas disponibles las cuales pueden dispararse y se agotan cuando se hace eso.

Basarse en los laboratorios anteriores.

Realizar el diagrama de clases de UML.

Tendrá 2 Ejércitos que pueden ser constituidos sólo por espadachines, caballeros y

arqueros (usar una estructura de datos por cada tipo de soldado). Cada ejército tendrá n soldados aleatorios entre 1 y 10. Cada soldado tendrá un nombre autogenerado: Espadachin0X1, Arquero1X1, Caballero2X2, etc., un valor de puntos de vida autogenerado aleatoriamente: Espadachín [3..4], Arquero [1..3] y Caballero [3..5], la fila y columna también autogenerados aleatoriamente (no puede haber 2 soldados en el mismo cuadrado) y valores autogenerados para el resto de atributos. Mostrar el tablero, distinguiendo los ejércitos y los tipos de soldados creados. Además, se debe mostrar todos los datos de todos los soldados creados para ambos ejércitos. También se mostrarán los datos del soldado con mayor vida de cada ejército, el promedio de puntos de vida de todos los soldados creados por ejército, los datos de todos los soldados por ejército en el orden que fueron creados y un ranking de poder de todos los soldados creados por ejército (del que tiene más nivel de vida al que tiene menos) usando algún algoritmo de ordenamiento. Finalmente, que muestre qué ejército ganará la batalla (indicar la métrica usada para decidir al ganador de la batalla). Hacerlo programa iterativo.

la clase main.java

-

```java
// RONI COMPANOCCA CHECCO
// CUI: 20210558
// LABORATORIO 16
// FUNDAMENTOS DE PROGRAMACION
import java.util.ArrayList;
import java.util.*;

public class main {

   public static void main(String[] args) {

      //DECLARACION DE VARIABLES Y ARREGLOS NECESARIOS
      Scanner scan = new Scanner(System.in);
      ArrayList<Soldado> ejercito1 = new ArrayList();
      ArrayList<Soldado> ejercito2 = new ArrayList();
      ArrayList<Soldado> ejercito3 = new ArrayList();
      ArrayList<Soldado> ejercito4 = new ArrayList();
      ArrayList<ArrayList<Soldado>> tablero = new ArrayList();
      int batallon1, batallon2, batallon3, batallon4, batallon5;
      int vidatotal1=0, vidatotal2=0, vidatotal3=0, vidatotal4=0;
      double promedioVida1=0, promedioVida2=0, promedioVida3=0, promedioVida4=0;

      // BUCLE PARA DESIGNAR LA CANTIDAD DE FILAS Y COLUMNAS DEL TABLERO
      for(int i=0; i<10; i++) {
         tablero.add(new ArrayList<Soldado>());
         for(int j=0; j<10; j++) {
            tablero.get(i).add(new Soldado());
         }
      }

      // CREACION DEL NUMERO DE POSICIONES DE CADA EJERCITO
      batallon1 = aleatorio(1,10);
      batallon2 = aleatorio(1,10);
      batallon3 = aleatorio(1,10);
      batallon4 = aleatorio(1,10);
      batallon5 = aleatorio(1,10);

      Ejercito eje = new Ejercito(batallon1, batallon2, batallon3, batallon4,
```

```
        batallon5);

    // INICIALIZAR ARREGLOS
    inicializarArreglo(ejercito1, batallon1);
    inicializarArreglo(ejercito2, batallon2);
    inicializarArreglo(ejercito3, batallon3);
    inicializarArreglo(ejercito4, batallon4);

    // GENERAR EJERCITOS VALIDOS
    generarEjercitos(ejercito1, ejercito2, 3, 4);
    generarEjercitos(ejercito3, ejercito4, 3, 4);

    //  AADIR  LOS EJERCITOS AL TABLERO
     aadirTablero(ejercito1, tablero);
     aadirTablero(ejercito2, tablero);
     aadirTablero(ejercito3, tablero);
     aadirTablero(ejercito4, tablero);

    // MOSTRAR OPCION PARA INGRESAR DATOS MANUALMENTE O AUTO GENERADO
    System.out.println("\n1-QUIERE INGRESAR LOS DATOS MANUALMENTE");
    System.out.println("\n2-QUIERE SUS DATOS DE MANERA AUTOGENERADO");
    String aux = scan.nextLine();
    int auxiliar = intTryParse(aux);
    if(auxiliar == 1) {
       // IMPRIMIR TODOS LOS DATOS DEL REYNO DE INGLATERRA
       Soldado eje1 = new Soldado("# REYNO DE INGLATERRA #");
       System.out.println("Cuantos Soldados vas a agregar");
       int sol = scan.nextInt();
       for( int i=0; i<sol; i++) {
          String nombre = scan.nextLine();
          int fila = scan.nextInt();
          int columna = scan.nextInt();
          int vida = scan.nextInt();
          eje1.addSoldado(new Soldado("Rodrigo",2,3,2));
       }
       // IMPRIMIR TODOS LOS DATOS DEL REYNO DE FRANCIA
       Soldado as = new Soldado("\n# REYNO DE FRANCIA #");
       System.out.println("Cuantos Soldados vas a agregar");
       int sol = scan.nextInt();
       for( int i=0; i<sol; i++) {
          String nombre = scan.nextLine();
          int fila = scan.nextInt();
          int columna = scan.nextInt();
          int vida = scan.nextInt();
          as.addSoldado(new Soldado("Rodrigo",2,3,2));
       }
        } else if(auxiliar == 2) {
       // IMPRIMIR TODOS LOS DATOS DEL REYNO DE INGLATERRA
       System.out.println("# REYNO DE INGLATERRA #");
       for(int i=0; i<ejercito1.size(); i++) {
          imprimir(ejercito1.get(i));
       }
       datos(ejercito1,vidatotal1,promedioVida1);
       // IMPRIMIR TODOS LOS DATOS DEL REYNO DE FRANCIA
       System.out.println("\n# REYNO DE FRANCIA #");
       for(int i=0; i<ejercito2.size(); i++) {
```

```
            imprimir(ejercito2.get(i));
        }
        datos(ejercito2,vidatotal2,promedioVida2);

        // IMPRIMIR TODOS LOS DATOS DEL REYNO DE SACRO IMPERIO
        System.out.println("\n# REYNO DE SACRO IMPERIO #");
        for(int i=0; i<ejercito3.size(); i++) {
            imprimir(ejercito3.get(i));
        }
        datos(ejercito3,vidatotal3,promedioVida3);
        // IMPRIMIR TODOS LOS DATOS DEL REYNO DE CASTILLA-ARAGON Y MOROS
        System.out.println("\n# REYNO DE CASTILLA-ARAGON Y MOROS #");
        for(int i=0; i<ejercito4.size(); i++) {
            imprimir(ejercito4.get(i));
        }
        datos(ejercito4,vidatotal4,promedioVida4);

        // IMPRIMIR LOS SOLDADOS ENFRENTADOS
              int suma = 0;
              Soldado mayor = new Soldado();
              Soldado max = new Soldado();
              mayor.setPuntos(0);
              for(int i=0; i<ejercito1.size();i++) {
                 if ((int)mayor.getPuntos()<ejercito1.get(i).getPuntos()) {
                    mayor = ejercito1.get(i);
                 }
              }
              for(int i=0; i<ejercito2.size();i++) {
                 if ((int)max.getPuntos()<ejercito2.get(i).getPuntos()) {
                    max = ejercito2.get(i);
                 }
              }
        // MOSTRAR PARA INICIAR EL JUEGO
        double porcentaje1 = 0, porcentaje2 = 0;
        Boolean intentos = true;
        while (intentos) {
            System.out.println("\nQUIERE EMPEZAR A JUGAR");
            System.out.println("1-SI\t 2-No");
            String reader = scan.nextLine();
            int opcion = intTryParse(reader);
            if(opcion == 1) {
               // ENFRENTAMIENTO DEL LOS REYNOS ALEATORIAMENTE
               int w=0, cantidad=4;
               int arreglo[] = new int[cantidad];

               arreglo[w]= aleatorio(1,4);
               for( w=1; w<cantidad ;w++) {
                  arreglo[w]= aleatorio(2,4);
                  for(int x=1; x<w; x++) {
                     if(arreglo[w]==arreglo[x]) {
                        w--;
                     }
                  }
               }
               for(int q=0; q<1; q++) {
                  if(arreglo[1] == 2) {
```

```
                System.out.println("\nREYNO DE INGLATERRA VS REYNO DE FRANCIA");
                imprimirTablero(tablero);
             }else if(arreglo[1] == 3) {
                System.out.println("\nREYNO DE INGLATERRA VS REYNO DE SACRO
                    IMPERIO");
                imprimirTablero(tablero);
             }else if(arreglo[1] == 4) {
                System.out.println("\nREYNO DE INGLATERRA VS REYNO DE
                    CASTILLA-ARAGON Y MOROS");
                imprimirTablero(tablero);
             }
          }
          for(int r=0; r<1; r++) {
             if(arreglo[2] == 2 & arreglo[3] == 3) {
                System.out.println("REYNO DE FRANCIA VS REYNO DE SACRO IMPERIO");
                imprimirTablero(tablero);
             }else if(arreglo[2] == 2 & arreglo[3] == 4) {
                System.out.println("REYNO DE FRANCIA VS REYNO DE CASTILLA-ARAGON
                    Y MOROS");
                imprimirTablero(tablero);
             }else if(arreglo[2] == 3 & arreglo[3] == 2) {
                System.out.println("REYNO DE SACRO IMPERIO VS REYNO DE FRANCIA");
                imprimirTablero(tablero);
             }else if(arreglo[2] == 3 & arreglo[3] == 4) {
                System.out.println("REYNO DE SACRO IMPERIO VS REYNO DE
                    CASTILLA-ARAGON Y MOROS");
                imprimirTablero(tablero);
             }else if(arreglo[2] == 4 & arreglo[3] == 2) {
                System.out.println("REYNO DE CASTILLA-ARAGON Y MOROS VS REYNO DE
                    FRANCIA");
                imprimirTablero(tablero);
             }else if(arreglo[2] == 4 & arreglo[3] == 3) {
                System.out.println("REYNO DE CASTILLA-ARAGON Y MOROS VS REYNO DE
                    SACRO IMPERIO");
                imprimirTablero(tablero);
             }
          }
          System.out.println("\nRonda: "+(1));
          suma = mayor.getPuntos()+max.getPuntos();
          System.out.println("Suma: "+suma);
          porcentaje1 = (100*mayor.getPuntos())/suma;
          porcentaje2 = (100*max.getPuntos())/suma;
          System.out.println("\nLa probabilidad de ganar del soldado del
              ejercito 1 es: "+porcentaje1+"%");
          System.out.println("La probabilidad de ganar del soldado del ejercito
              2 es: "+porcentaje2+"%");
          break;
       } else if(opcion == 2) {
          System.out.println("QUIERE SALIR DE JUEGO");
          System.out.println("1-SI\t 2-No");
          if(!(fin())) {intentos = false;}
       }
    }
    // PRIMERA METRICA PARA DECIDIR EL GANADOR
    if(porcentaje1>porcentaje2) {
       System.out.println("\nGANADOR ***EJERCITO 1***");
```

```
        }else if (porcentaje1<porcentaje2) {
           System.out.println("\nGANADOR ***EJERCITO 2***");
           }
        else {
           System.out.print("\n***ES UN EMPATE***");
        }
        // SEGUNDA METRICA PARA DECIDIR EL GANADOR
        if(porcentaje1>porcentaje2) {
           System.out.println("\nGANADOR ***EJERCITO 1***");
        }else if (porcentaje1<porcentaje2) {
           System.out.println("\nGANADOR ***EJERCITO 2***");
           }
        else {
           System.out.print("\n***ES UN EMPATE***");
        }
        // TERCERA METRICA PARA DECIDIR EL GANADOR
        if(vidatotal1<vidatotal2) {
           System.out.println("\nGANADOR ***EJERCITO 1***");
        }else if (vidatotal1<vidatotal2) {
           System.out.println("\nGANADOR ***EJERCITO 2***");
           }
        else {
           System.out.print("\n***ES UN EMPATE***");
        }
     }
  }

  // METODO PARA SALIR
  public static Integer intTryParse(String num) {
     try {
        return Integer.parseInt(num);
     }catch (NumberFormatException e) {
        return 0;
     }
  }
  public static Boolean fin() {
     Scanner scan = new Scanner(System.in);
     Boolean intentos = true;
     while (intentos) {
        String reader = scan.nextLine();
        int opcion = intTryParse(reader);
        if(opcion == 1) {
           intentos = false;
           break;
        } else if(opcion == 2) {
           intentos = true;
           break;
        }
     }
     return intentos;
  }
  // METODO PARA MOSTAR LOS DATOS
  public static void datos(ArrayList<Soldado>M2, int a, double pro) {
     ArrayList<Soldado>Soldados = new ArrayList();
     ordenarPorVidaDescendente(M2);
     System.out.println("\nOrdenados por nivel de vida");
```

```
   for(int i=0; i<M2.size(); i++) {
      imprimir(M2.get(i));
   }
   //IMPRIMIR LA VIDA TOTAL Y EL PROMEDIO DEL EJERCITO
   System.out.println("\nVida total y el promedio del ejercito: ");
   for (int i=0; i<M2.size(); i++) {
      a+=M2.get(i).getPuntos();
      pro = a/(M2.size()*1.0);
   }
   System.out.println("Vida total: "+a);
   System.out.println("Promedio de vida: "+pro);
}

// METODO PARA CREAR NUMEROS ALEATORIOS EN UN RANGO
public static int aleatorio(int min, int max) {
   return(int)(Math.random()*(max-min+1)+min);
}

// METODO PARA INICIAR UN ARRAYLIST
public static void inicializarArreglo (ArrayList<Soldado> soldadito, int num) {
   for (int i=0; i<num; i++) {
      soldadito.add(new Soldado());
   }
}

// METODO PARA GENERAR DATOS DEL OBJETO SOLDADO
public static Soldado generarDatos() {
   Soldado soldadito = new Soldado();
   soldadito.setPuntos(aleatorio(1,10));
   soldadito.setColumna(aleatorio(1,10));
   soldadito.setFila(aleatorio(1,10));
   return soldadito;
}

// METODOS PARA GENERAR LOS EJERCITOS DE MANERA ALEATORIA
   public static void generarEjercitos(ArrayList<Soldado>B1,
       ArrayList<Soldado>B2, int a, int b) {
      ArrayList<Soldado>Soldados = new ArrayList();
      Soldados.add(generarDatos());
      for (int i=1; i<(B1.size()+B2.size()); i++) {
         Soldados.add(generarDatos());
         for (int j=0; j<i; j++) {
            if(Soldados.get(i).getFila()==Soldados.get(j).getFila()) {
               if(Soldados.get(i).getColumna()==Soldados.get(j).getColumna()){
                  Soldados.remove(i);
                  i--;
               }
            }
         }
      }
      for (int i=0; i<B1.size(); i++) {
         B1.add(i, Soldados.get(i));
         B1.get(i).setNombre("Soldado"+i+"x"+a);
         B1.get(i).setColumn(B1.get(i).getPuntos()+"[E"+a+"]");
         B1.remove(i+1);
      }
```

```
      for (int i=0; i<B2.size(); i++) {
         B2.add(i, Soldados.get(i+B1.size()));
         B2.get(i).setNombre("Soldado"+i+"x"+b);
         B2.remove(i+1);
         B2.get(i).setColumn(B2.get(i).getPuntos()+"[E"+b+"]");
      }
   }


// METODO PARA AADIR LOS EJERCITOS AL TABLERO
public static void aadirTablero(ArrayList<Soldado>soldadito,
    ArrayList<ArrayList<Soldado>>table) {
   for (int i=0; i<soldadito.size(); i++) {
      table.get(soldadito.get(i).getColumna()-1).add(soldadito.get(i).getFila()-1,soldadito.get(i));
      table.get(soldadito.get(i).getColumna()-1).remove(soldadito.get(i).getFila());
   }
}

// METODO PARA IMPRIMIR EL TABLERO EN LA CUAL SE DESARROLLA EL JUEGO
public static void imprimirTablero(ArrayList<ArrayList<Soldado>> table) {
   System.out.println("\tA\tB\tC\tD\tF\tG\tH\tI\tJ");
   for(int i=0; i<table.size(); i++) {
      System.out.print(i+1);
      for(int j=0; j<table.get(i).size();j++) {
         System.out.print("\t"+table.get(i).get(j).getColumn()+"|");
      }
      System.out.println("\n");
   }
}

//METODO PARA IMPRIMIR LOS SOLDADOS DE MAYOR VIDA
public static Soldado SoldadoConMayorVida (ArrayList<Soldado>soldadito) {
   Soldado mayor = new Soldado();
   mayor.setPuntos(0);
   for(int i=0; i<soldadito.size();i++) {
      if (mayor.getPuntos()<soldadito.get(i).getPuntos()) {
         mayor = soldadito.get(i);
      }
   }
   return mayor;
}
//METODO PARA IMPRIMIR LOS SOLDADOS DE MAYOR VIDA PERO SOLO SUS PUNTOS DE VIDA
public static void SoldadoMayorVida (ArrayList<Soldado>soldadito) {
   Soldado mayor = new Soldado();
   mayor.setPuntos(0);
   for(int i=0; i<soldadito.size();i++) {
      if (mayor.getPuntos()<soldadito.get(i).getPuntos()) {
         mayor = soldadito.get(i);
      }
   }
   leer(mayor);
}

// METODO PARA IMPRIMIR EL NOMBRE, LA POSICION Y NIVEL DE VIDA DEL SOLDADO
public static void imprimir(Soldado soldadito) {
   System.out.println("Nombre: "+soldadito.getNombre()+"\nPosicion:
```

```
            "+soldadito.getColumna()+"X"+soldadito.getFila()+"\tVida:
            "+soldadito.getPuntos());
    }
    // METODO PARA IMPRIMIR EL NOMBRE, LA POSICION Y NIVEL DE VIDA DEL SOLDADO
    public static void leer(Soldado soldadito) {
       System.out.println(soldadito.getPuntos());
    }

    // METODO QUE NOS AYUDA A ORDENAR LOS SOLDADOS DE ACUERDO A SU NIVEL DE VIDA
    public static void ordenarPorVida(ArrayList<Soldado>soldadito) {
       Soldado aux = new Soldado();
       for(int i=0; i<soldadito.size()-1; i++) {
          for(int j=0; j<soldadito.size()-i-1; j++) {
             if(soldadito.get(j).getPuntos()>soldadito.get(j+1).getPuntos()) {
                aux = soldadito.get(j);
                soldadito.set(j,soldadito.get(j+1));
                soldadito.set(j+1,aux);
             }
          }
       }
    }
    // METODO QUE NOS AYUDA A ORDENAR LOS SOLDADOS DE ACUERDO A SU NIVEL DE VIDA (DE
        MAYOR A MENOR)
    public static void ordenarPorVidaDescendente(ArrayList<Soldado>soldadito) {
       Soldado aux = new Soldado();
       for(int i=0; i<soldadito.size()-1; i++) {
          for(int j=0; j<soldadito.size()-i-1; j++) {
             if(soldadito.get(j).getPuntos()<soldadito.get(j+1).getPuntos()) {
                aux = soldadito.get(j);
                soldadito.set(j,soldadito.get(j+1));
                soldadito.set(j+1,aux);
             }
          }
       }
    }

}
```

- la clase Soldado.java

```
// RONI COMPANOCCA CHECCO
// CUI: 20210558
// LABORATORIO 12
// FUNDAMENTOS DE PROGRAMACION
// CLASE SOLDADO PARA LOS METODOS SETTER Y GETTER
public class Soldado {
    private String nombre;
    private int nivelAtaque;
    private int nivelDefensa;
    private int nivelVida;
    private int vidaActual;
    private int velocidad;
    private String actitud;
    private boolean vive;

    public Soldado() {
```

```
        nombre = "";
        nivelAtaque = 0;
        nivelDefensa = 0;
        nivelVida = 0;
        vidaActual = nivelVida;
        velocidad = 0;
        actitud = "";
        vive = true;
    }

    public void atacar(Soldado enemigo) {
        int danio = this.nivelAtaque - enemigo.nivelDefensa;
        if (danio > 0) {
            enemigo.serAtacado(danio);
            System.out.println(this.nombre + " atac a " + enemigo.nombre + " y le
                caus " + danio + " de dao.");
        } else {
            System.out.println(this.nombre + " atac a " + enemigo.nombre + " pero no
                le caus dao.");
        }
    }

    public void defender() {
        // Lgica para la defensa
        // Puede disminuir el dao recibido en un futuro ataque, por ejemplo
        this.nivelDefensa += 10; // Aumentar la defensa por ejemplo
        System.out.println(this.nombre + " se est defendiendo.");
    }

    public void avanzar() {
        // Lgica para avanzar en el juego
        // Podra mover al soldado a una nueva posicin en el tablero, por ejemplo
        if (this.velocidad > 0) {
            // Mover al soldado en la direccin correspondiente
            System.out.println(this.nombre + " est avanzando.");
        } else {
            System.out.println(this.nombre + " no puede avanzar, la velocidad es
                0.");
        }
    }

    public void retroceder() {
        // Lgica para retroceder en el juego
        // Podra mover al soldado hacia atrs en el tablero, por ejemplo
        System.out.println(this.nombre + " est retrocediendo.");
    }

    public void serAtacado(int danioRecibido) {
        this.vidaActual -= danioRecibido;
        if (this.vidaActual <= 0) {
            this.morir();
        }
    }

    public void huir() {
        // Lgica para huir del combate
```

```
        // Podra cambiar la posicin del soldado a una zona segura, por ejemplo
        if (this.nivelVida < 10) {
            // Huir solo si la vida es baja
            System.out.println(this.nombre + " est huyendo del combate.");
        } else {
            System.out.println(this.nombre + " no puede huir, su vida est alta.");
        }
    }

    public void morir() {
        this.vidaActual = 0;
        this.vive = false;
    }

    public void setVidaActual(int vidaActual) {
        this.vidaActual = vidaActual;
    }

    public int getVidaActual() {
        return vidaActual;
    }

    // Getters y Setters

    public String getNombre() {
        return nombre;
    }

    public String getNivelVida() {
        return nivelVida;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getNivelAtaque() {
        return nivelAtaque;
    }

    public int setNivelVida() {
        this.nivelVida = nivelVida;
    }

    public void setNivelAtaque(int nivelAtaque) {
        this.nivelAtaque = nivelAtaque;
    }

    public int getNivelDefensa() {
        return nivelDefensa;
    }

    public void setNivelDefensa(int nivelDefensa) {
        this.nivelDefensa = nivelDefensa;
    }
```

```
    public boolean isVive() {
        return vive;
    }

    public void setVive(boolean vive) {
        this.vive = vive;
    }
}
```

- la clase Ejercito.java

```
import java.util.ArrayList;

public class Ejercito {
    private String reino;
    private ArrayList<Soldado> misSoldados;

    // Constructor
    public Ejercito(String reino) {
        this.reino = reino;
        this.misSoldados = new ArrayList<>();
    }

    // Getters y Setters
    public String getReino() {
        return reino;
    }

    public void setReino(String reino) {
        this.reino = reino;
    }

    // Mtodo para agregar un soldado al ejrcito
    public void agregarSoldado(Soldado soldado) {
        if (misSoldados.size() < 10) {
            misSoldados.add(soldado);
        } else {
            System.out.println("El ejrcito ya tiene el mximo de soldados permitidos
                (10).");
        }
    }

    // Mtodo para eliminar un soldado por nombre
    public void eliminarSoldado(String nombreSoldado) {
        for (Soldado soldado : misSoldados) {
            if (soldado.getNombre().equals(nombreSoldado)) {
                misSoldados.remove(soldado);
                System.out.println("Soldado eliminado exitosamente: " +
                    nombreSoldado);
                return;
            }
        }
        System.out.println("Soldado no encontrado: " + nombreSoldado);
    }

    // Mtodo para modificar un atributo de un soldado por nombre
```

```
    public void modificarSoldado(String nombreSoldado, String atributo, Object
        valor) {
        for (Soldado soldado : misSoldados) {
            if (soldado.getNombre().equals(nombreSoldado)) {
                switch (atributo) {
                    case "nombre":
                        soldado.setNombre((String) valor);
                        break;
                    // Agregar ms casos segn los atributos que se puedan modificar
                    default:
                        System.out.println("Atributo no vlido: " + atributo);
                        return;
                }
                System.out.println("Soldado modificado exitosamente: " +
                    nombreSoldado);
                return;
            }
        }
        System.out.println("Soldado no encontrado: " + nombreSoldado);
    }

    // Mtodo para consultar el soldado con mayor nivel de ataque
    public Soldado consultarSoldadoMasPoderoso() {
        Soldado masPoderoso = null;
        int maxNivelAtaque = Integer.MIN_VALUE;

        for (Soldado soldado : misSoldados) {
            if (soldado.getNivelAtaque() > maxNivelAtaque) {
                maxNivelAtaque = soldado.getNivelAtaque();
                masPoderoso = soldado;
            }
        }

        return masPoderoso;
    }

    // Mtodo para ver el ranking de poder considerando el nivel de vida
        (descendente)
    public ArrayList<Soldado> consultarRankingPoder() {
        ArrayList<Soldado> ranking = new ArrayList<>(misSoldados);
        ranking.sort((s1, s2) -> Integer.compare(s2.getVidaActual(),
            s1.getVidaActual()));
        return ranking;
    }

    // Mtodo para ver todos los datos del ejrcito y de los soldados que lo
        conforman
    @Override
    public String toString() {
        StringBuilder result = new StringBuilder("Ejrcito de " + reino + ":\n");
        for (Soldado soldado : misSoldados) {
            result.append("\t").append(soldado.toString()).append("\n");
        }
        return result.toString();
    }
}
```

- Ejecucion

```
          A      B      C      D      F      G      H      I      J
1                                          2[E1]
2                                                        5[E1]
3
4
5
6                                                                 2[E2]
7
8                            2[E1]  3[E2]
9       3[E2]                                                     2[E1]
10                                         2[E1]          3[E2]
```

PROBLEMA 02

Basándose en los laboratorios anteriores

Crear la clase Mapa, que esté constituida por el tablero antes visto, pero que en lugar de soldados, posicione ejércitos en ciertas posiciones aleatorias (entre 1 y 10 ejércitos por cada reino). Se deben generar ejércitos de 2 reinos, entre 1 y 10 soldados por ejército. No se admite guerra civil. El Mapa tiene como atributo el tipo de territorio que puede ser (bosque, campo abierto, montaña, desierto, playa). La cantidad de ejércitos y cantidad de soldados por ejército, así como todos sus atributos se deben generar aleatoriamente

Dibujar el Mapa con las restricciones que sólo 1 ejercito como máximo en cada cuadrado

El mapa tiene un solo tipo de territorio autogenerado

Considerar que el territorio influye en los resultados de las batallas, así cada reino tiene bonus según el territorio: Inglaterra-¿bosque, Francia-¿campo abierto, Castilla-Aragón-¿montaña, Moros-¿desierto, Sacro Imperio RomanoGermánico-¿bosque, playa y campo abierto. En dichos casos, se aumenta el nivel de vida en 1 a todos los soldados del reino beneficiado

Realizar el diagrama de clases de UML completo

Indicar quién ganaría la guerra y justificar por qué. Describir 3 métricas usadas

Hacerlo iterativo

Ejecucion

-
```
|      Ejrcito      |
+-------------------+
| - reino: String   |
```

```
| - misSoldados: ArrayList<Soldado> |
+---------------------+
| + Ejrcito(reino: String) |
| + setReino(reino: String) |
| + getReino(): String   |
| + agregarSoldado(soldado: Soldado) |
| + eliminarSoldado(nombre: String) |
| + modificarSoldado(nombre: String, atributo: String, valor: Object) |
| + consultarSoldadoMasPoderoso(): Soldado |
| + consultarRankingPoder(): ArrayList<Soldado> |
| + toString(): String    |
+---------------------+

+---------------------+
|     Soldado       |
+---------------------+
| - nombre: String   |
| - nivelAtaque: int |
| - nivelDefensa: int |
| - puntos: int      |
| - vidaActual: int  |
| - velocidad: int   |
| - actitud: String  |
| - vive: boolean    |
+---------------------+
| + Soldado(nombre: String, nivelAtaque: int, nivelDefensa: int, puntos: int,
    vidaActual: int, velocidad: int, actitud: String) |
| + setNombre(nombre: String) |
| + setNivelAtaque(nivelAtaque: int) |
| + setNivelDefensa(nivelDefensa: int) |
| + setPuntos(puntos: int) |
| + setVidaActual(vidaActual: int) |
| + setVelocidad(velocidad: int) |
| + setActitud(actitud: String) |
| + setVive(vive: boolean) |
| + serAtacado(danio: int) |
| + atacar() |
| + avanzar() |
| + defender() |
| + huir() |
| + retroceder() |
| + morir() |
| + aumentarVida() |
| + toString(): String |
+---------------------+
       Ejercito
|- reino: String
|- misSoldados: ArrayList<Soldado>

|+ Ejercito(reino: String)
|+ getReino(): String
|+ setReino(reino: String): void
|+ agregarSoldado(soldado: Soldado): void
|+ eliminarSoldado(nombreSoldado: String): void
|+ modificarSoldado(nombreSoldado: String, atributo: String, valor: Object): void
|+ consultarSoldadoMasPoderoso(): Soldado
```

```
|+ consultarRankingPoder(): ArrayList<Soldado>
|+ toString(): String
```

6. CUESTIONARIO

- ¿Para que usamos la herencia y las jerarquías de clases?
  La herencia permite la creación de una nueva clase basada en una clase existente, lo que permite reutilizar el código de la clase base. Esto reduce la duplicación de código y facilita el mantenimiento.
- ¿Qué es el enlace dinámico?
  El enlace dinámico, también conocido como enlace dinámico o vinculación dinámica, es un proceso en el que ciertas operaciones y referencias a funciones o bibliotecas externas se resuelven durante el tiempo de ejecución en lugar del tiempo de compilación. Esto implica que las direcciones de memoria y las referencias a funciones no se determinan hasta que el programa se está ejecutando.
- ¿Qué ventajas tiene el polimorfismo?
  El polimorfismo permite que un objeto sea tratado como un objeto de su clase base, lo que brinda flexibilidad al código. Esto facilita la adaptabilidad del sistema a cambios y nuevas adiciones sin afectar el código existente.
- ¿Qué tipos de polimorfismo existen?
  Existen dos tipos principales de polimorfismo en programación orientada a objetos: el polimorfismo de compilación (o estático) y el polimorfismo de ejecución (o dinámico).

7. REFERENCIAS

- M. Aedo, "Fundamentos de Programación 2 - Tópicos de Programación Orientada a Objetos", Primera Edición, 2021, Editorial UNSA.
- `https://github.com/rescobedoq/programacion.git`
- J. Dean, Ïntroduction to programming with Java: A Problem Solving Approach", Third Edition, 2021, McGraw-Hill.
- C. T. Wu, .ªn Introduction to Object-Oriented Programming with Java", Fifth Edition, 2010, McGraw-Hill.
- P. Deitel, "Java How to Program", Eleventh Edition, 2017, Prentice Hall.